NPSCS-92-017

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Prototyping Hard Real-Time Ada Systems in a Classroom Environment

Luqi, M. Shing, P. Barnes and G. Hughes

Approved for public release; distribution is unlimited.

Prepared for:

Naval Postgraduate School
Monterey, California 93943

NAVAL POSTGRADUATE SCHOOL

Monterey, California

Rear Admiral R.W. West, Jr.
Superintendent

HARRISON SHULL
Provost

This report was prepared with research funded by the Naval Research Funds provided by the Naval Postgraduate School.

This report was prepared by:

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPSCS-92-017	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School	6b. OFFICE SYMBOL *(if applicable)* CS
7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943	7b. ADDRESS *(City, State, and ZIP Code)*
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Naval Postgraduate School	8b. OFFICE SYMBOL *(if applicable)*
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Prototyping Hard Real-Time Ada Systems in a Classroom Environment

12. PERSONAL AUTHOR(S)
Luqi, M. Shing, P. Barnes and G. Hughes

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
	FROM _____ TO _____	1992, December, 31	20

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Teaching graduate students how to develop hard real-time Ada software for embedded systems is a challenging task. We successfully used Ada in a series of software engineering courses to teach graduate students the characteristics of hard real-time software and fundamental skills to develop and validate complex systems and timing requirements through software prototypes of the systems. A research tool, called CAPS (Computer Aided Prototyping System)*, was used by the student software designers to construct software prototypes based on the requirements of the system as well as to automatically generate Ada code interconnecting reusable modules. The approach greatly stimulated the students' interest and helped them to gain first hand experiences in developing hard real-time systems.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Luqi	22b. TELEPHONE *(Include Area Code)* (408) 656-2735	22c. OFFICE SYMBOL CSLq

D FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Prototying Hard Real-Time Ada Systems in a Classroom Environment

Luqi, M. Shing, P. Barnes and G. Hughes
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

ABSTRACT

Teaching graduate students how to develop hard real-time Ada software for embedded system is a challenging task. We successfully used Ada in a series of software engineering courses to teach graduate students the characteristics of hard real-time software and fundamental skills to develop and validate complex systems and timing requirements through software prototypes of the systems. A research tool, called CAPS (Computer Aided Prototyping System)*, was used by the student software designers to construct software prototypes based on the requirements of the system as well as to automatically generate Ada code interconnecting reusable modules. The approach greatly stimulated the students' interest and helped them to gain first hand experiences in developing hard real-time systems.

*This research was supported in part by the National Science Foundation under grant numberCCR9058453.

KEYWORDS

Embedded Systems, Software Prototyping, Computer Aided Software Engineering, Ada Education, Hard Real-Time Systems, Requirements Engineering.

1. INTRODUCTION

Many embedded software systems are safety critical, and depend on meeting hard real-time deadlines for their successful operation. Patriot missile control software is one example of this kind of system. "On February 25, 1991, a Patriot missile defense system operating at Dhahran, Saudi Arabia, during Operation Dessert Storm failed to track and intercept an incoming Scud. This Scud subsequently hit an Army barracks, killing 28 Americans" [1].

The single most costly event in the Operation Dessert Storm was the result of a software timing error. It underscores the importance and difficulties in the design and development of hard real-time software in our military systems. Hard real-time systems are defined as those software systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced [2, 3]. One of the major differences between a hard real-time system and a conventional system is that the application software must meet its deadlines even under worst case conditions. Hard real-time software systems are typically embedded in larger systems, performing critical control functions. These real-time control functions require the software system to interact with a wide variety of hardware/software subsystems via networks. The process of design and development of these systems is often plagued with uncertainty, ambiguity and inconsistency. Typical tactical control software consists of numerous processes including communication processing, known-object recognition, track pro-

cessing, and display processing, among others. While conceptually these processes could frequently run concurrently, limitations in hardware resources often force these processes to be sequentialized in a centralized implementation through an interrupt driven prioritization scheme. This scheme produces a completely dynamic schedule whose effects are difficult to predict and control. The timing requirements are difficult for the user to provide and for the analysts to determine. As the software is modified, various aspects of its execution behavior change, including maximum execution times and execution precedences for the subfunctions. Often these changes are only observed after the fact: the system crashes during testing, or required functions don't get processed when needed. The response is often to tweak the schedule until the system appears to run acceptably on the available test cases. There is no assurance that the system will perform acceptably in situations that have not been explicitly tested, which usually includes conditions that may arise during the actual operation of the system.

A better approach to such problems involves systematic and automatable methods for constructing schedules and real-time simulations (or prototypes) of the systems. The Computer Aided Prototyping System (CAPS) is a research tool developed at the Naval Postgraduate School to provide an integrated software development environment aimed at rapidly prototyping hard real-time embedded software systems and generating Ada code automatically [4]. CAPS has undergone extensive testing by several classes of graduate students.

Teaching graduate students how to develop hard real-time Ada software for embedded system is a challenging task. Software Engineering is a fast moving discipline. Utilizing the state of the art research results for teaching graduate courses is an important supplemental way to ensure the quality of the courses. Such a practice has been instrumental in refining systematical approaches to requirements analysis [5]. Due to lack of good textbooks, we had to use the same approach to create our own teaching materials based on our research results and tools. As a result, we successfully used Ada in a series of software engineering courses to teach graduate students the characteristics of hard real-time software and fundamental skills to develop and validate complex systems and timing requirements through software prototypes of the systems to accomplish the important learning goals of our graduate students for their future DoD system acquisition tasks [6].

The series of Software Engineering courses employing Ada includes "Software Methodology"(CS3460) as the introductory course for the computer science majors, "Software Development for Combat Systems"(CS3050) as the introductory course for the non-computer science majors, followed by "Software Engineering"(CS4500), "Advanced Software Engineering"(CSS4520), and "Software Engineering with Ada"(CS4530). We developed an advanced textbook, *Software Engineering with Abstractions - An Integrated Approach for Large Ada Software*, to support this series of courses with a combination of formal approaches, a second-order logic for specifications of concurrent and real-time systems, and detailed examples of practical applications [7]. This textbook is used in CS4500 to give solid graduate education in Software Engineering to the computer science majors. Advanced technology, research topics, theoretical issues and skills for software automation were taught in CS4520 and CS4530. At the beginning of the series, for the non-computer science majors or the graduate students who have never experienced software system development, a fast and practical exploration was needed before they could possibly understand why the rigorous formalism for system specifications was needed in the textbook [7].

In order for the students to gain first hand experience in software development, they are required to work on the large projects in CS3460 or CS3050. In the courses, the student designers

used CAPS to construct software prototypes based on the requirements of the system and to automatically generate Ada code interconnecting reusable modules. Hardware components and interfaces were simulated in the CAPS prototype model. The approach greatly stimulated the students' interest and helped them to gain first hand experiences in developing hard real-time systems in Ada. The students in these courses designed and implemented prototypes of several hard real-time systems: the *Patriot Missile Defense System*, the *Robot Motion Control System*, the *Fish Farm Control System* and the *Generic Command and Control Station*. This paper summarizes the experience gained from these projects.

2. THE COMPUTER AIDED PROTOTYPING SYSTEM

CAPS provides facilities for computer-aided design, software component reuse, and automated Ada code generation (Figure 1). These tools are developed to help software engineers rapidly construct and adapt software, validate and refine user requirements, and to check consistency of proposed design [8]. The process supported by CAPS provides requirements and designs in a form that is useful in the construction of the operational system, as well as in the acquisition process to assess optimized implementations delivered by contractors, and to integrate independently developed subsystems. CAPS allows the user to specify the requirements of a proposed system informally in a structured top-down fashion using easy-to-understand graphics, assists the designer in augmenting the informal specifications with formal annotations, and automatically translates the result into executable Ada code that realizes the timing requirements based on declared maximum execution times and monitors conformance of actual execution times to the maximum execution times specified in the design.

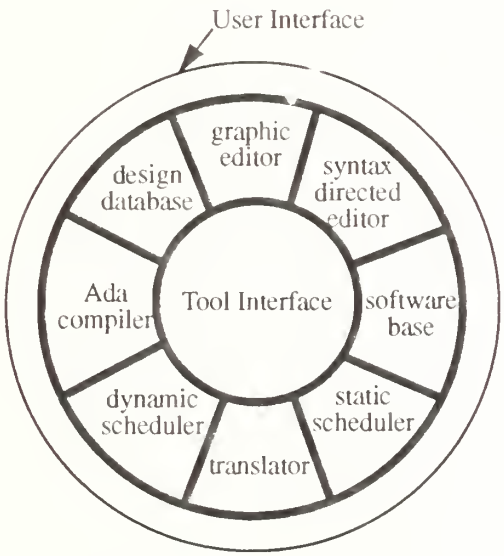


Figure 1. CAPS Advanced Rapid Prototyping Environment

2.1 THE CAPS METHOD

There are four major stages in the CAPS rapid prototyping process: software system design, construction, execution, and debugging/modification (Figure 2). The initial prototype design starts with an analysis of the problem and a decision about which parts of the proposed system are

to be prototyped. Requirements for the prototype are then generated, either informally (e.g. English) or in some formal notation. These requirements may be refined by asking users to verify their completeness and correctness. After the requirements analysis is completed, the designer uses the CAPS graphic editor to draw dataflow diagrams with nonprocedural timing and control constraints as part of the specification of a hierarchically structured prototype, resulting in a preliminary, top-level design free from programming level details. The underlying computational model unifies dataflow and control flow, and provides a mechanism for developing top-down decompositions. The user may continue to decompose any software module until its components can be realized via reusable components drawn from the software base or new atomic components. This high-level specification is then translated into Ada for execution and evaluation. Debugging and modification is assisted by a design database that helps the designers in managing the design history and coordinating changes.

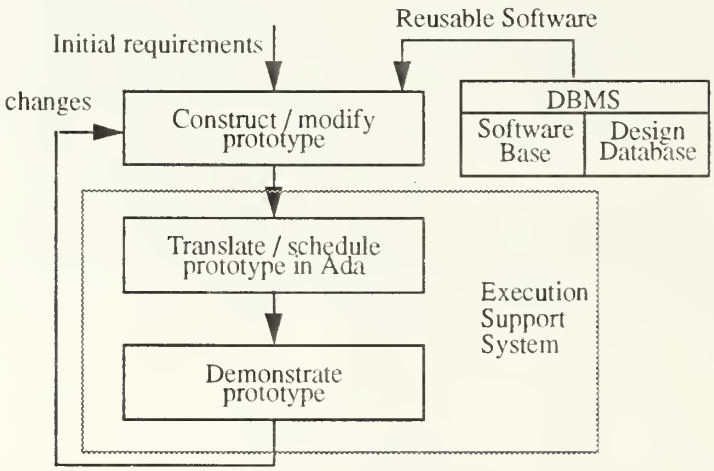


Figure 2. Iterative Prototyping Process in CAPS

2.2 THE PROTOTYPE SYSTEM DESCRIPTION LANGUAGE

The CAPS tools are based on the Prototype System Description Language (PSDL). PSDL is a high-level language designed specifically to support the specification of real-time software systems, as well as to organize and retrieve reusable components in the software base [9]. PSDL lets designers sketch a system using computation graphs, where the vertices are operators and the directed edges are data streams, and then refine the design by adding timing and control constraints in text form.

Operators are state machines whose internal states are modeled by state variables. Operators with an empty variable set behave like functions. PSDL operators can be triggered by data (*sporadic* operators) or by periodic timing constraints (*periodic* operators). When triggered, an operator will produce output based on input values and values of internal state variables. There are two kinds of operators: *atomic* operators and *composite* operators. An atomic operator is one that can be realized by an implementation stored in the software base or supplied by the software engineers, and a composite operator is one that can be decomposed into a network of more primitive operators represented as enhanced dataflow diagrams.

Operators communicate via two kinds of data streams: *dataflow* streams and *sampled*

streams. A dataflow stream can be thought of as a FIFO buffer of capacity one that connects synchronized operators. Data in dataflow streams represents discrete transactions, and is removed from the stream when read. A sampled stream can be thought of as a single memory cell and connects operators with uncoordinated rates. This type of data usually comes from a continuous data source and can be used many times or written over before use, depending on the rate of its input and use.

Real-time applications, design flexibility, and code reuse motivate the timing and non-procedural control constraints of PSDL. Each time critical operator has a *maximum execution time* constraint, representing the maximum time the operator may need to complete execution after it is fired, given access to all required resources. In addition, each periodic operator has a *period* and a *deadline*. The period is the interval between triggering times for the operator and the deadline is the maximum duration from the triggering of the operator to the completion of its operation. Each sporadic operator has a *maximum response time* and a *minimum calling period*. The minimum calling period is the smallest interval allowed between two successive triggerings of a sporadic operator. The maximum response time is the maximum duration allowed from the triggering of the sporadic operator to the completion of its operation. To model distributed systems, PSDL also provides the option of specifying the *maximum delay* associated with any data stream.

PSDL also allows the specification of both input and output guards to provide conditional execution of an operator and conditional output of data. Guards can include conditions on timers that measure durations of system states, and can allow operators to execute only when fresh data has been written to an input stream. Control constraints can also trigger exceptions.

2.3 THE CAPS TOOLS

The set of tools provided by CAPS includes the user interface, software database system, and execution support system.

The user interface in CAPS includes a graphic editor, a syntax-directed editor, and a browser. The graphic editor and the syntax-directed editor together provide a user-friendly environment for the user/software engineer to construct a prototype using a combination of graphical and textual objects. The browser allows software engineers to view reusable components in the software database system.

The software database system, which consists of a software base and a design database, provides facilities for software reuse, automated project management, and version control. The software base keeps track of the PSDL descriptions and Ada implementations for all reusable software components in CAPS. The design database coordinates the concurrent efforts of a team of software engineers and manages the different versions and alternatives of the design and documents they produce.

The execution support system consists of a translator, a static scheduler and a dynamic scheduler. The translator generates code that binds together the code supplied by the designer and the reusable components extracted from the software base. The static scheduler and the dynamic scheduler together create the real-time schedule and control code needed for executing the prototype. The resultant Ada main program consists of four parts. The first is a set of data streams, implemented as instantiation of generic packages containing Ada tasks controlling the mutually exclusive read/write access to the data buffers. The second part consists of a set of drivers, one for each of the atomic operators. Each driver reads data from the specified input streams, evaluates the input guards, executes the operators, evaluates the output guards and then writes the outputs to

the specified data streams accordingly. The third part is a static schedule which is a high priority Ada task that executes all time-critical operators in a deterministic and timely manner. The schedule is generated automatically based on the timing constraints and the precedence of the operators specified in the data-flow graph. The last part of the Ada main program is a dynamic schedule, which is a low priority Ada task that executes the non-time-critical operators during the slack time in the static schedule.

3. OUR CLASSROOM EXPERIMENTS

Four hard real-time software system prototypes have been designed and developed by students using CAPS. Due to lack of space, we shall only present the *Patriot Missile Defense System* and the *Generic Command and Control Station* prototypes here.

3.1 THE PATRIOT MISSILE DEFENSE SYSTEM

Figure 3 shows the PSDL specification of the prototype for a simplified two-dimensional version of the Patriot missile defense system. This prototype was developed by five Weapons System students for the class "Software Development for Combat Systems." In this course, non-computer-science students learned the basics of real-time systems, software engineering, and real-time Ada programming. They also must learn to use UNIX workstations running X Window Systems and Motif along with software tools such as Verdix Ada Development System (VADS), CAPS and TAE Plus. The prototype shown in Figure 3 has a total of nine atomic operators. The operators *patriot_radar, check_threat, control_patriort* and *scud_radar* are time-critical, each has a maximum execution time of 10 ms.

Figure 4 shows the TAE Plus user interface for the prototype, which was developed using the TAE Work Bench and consists of two panels, one representing the Scud launcher and a tracking radar, and the other representing the Patriot System Console. Two major problems had to be overcome to use TAE Plus user interfaces for the CAPS prototypes. First, the user interface must be subordinate to the real-time system (i.e. execution of the time-critical operators cannot be blocked by the user interface processing), while the architecture generated by TAE Plus is one where the user interface is in control, executing application functions as driven by user actions. Second, the X Window System Xlib is non-reentrant. Thus user interface operations may not be interrupted by other user interface operations before completion. Our solution to this problem involved a user interface architecture where all calls to the user interface by the real-time applications are made mutually exclusive through an Ada monitor task. All rendezvous are embodied in a selective wait with a 10 ms time-out. If no rendezvous are ready within the 10 ms, the task exits the selective wait and checks for TAE WPT events. All WPT events are processed before checking for rendezvous requests again. (Interested readers should refer to [10] for details.)

3.2 GENERIC COMMAND AND CONTROL STATION

This prototype was developed by a group of five Computer Science students for a class created by the first author titled "Software Engineering with Ada." In this course, students learn how to apply the software engineering principles in the design of large, real-time, embedded computer systems through the use of automated tools in the Ada environment. The class project was to build upon previous prototyping efforts on a SUN3 workstation [11] and use the CAPS tools to develop an improved prototype for a generic C3I station and simulations of its interacting external systems on the SPARCstations. Figure 5 shows the PSDL specification of the top-level design of the pro-

totype, which has four composite operators (*comms_interface, track_database_manager, user_interface* and *sensor_interface*) and five atomic operators (*comms_links, weapons_systems, navigation_system, sensors* and *weapons_interface*). The operators *comms_links, weapons_systems, navigation_system* and *sensors* are time-critical software modules which simulate the external systems with which the C3I station interacts.

Figures 6 and 7 show the PSDL specifications of the successive decomposition of the *comms_interface* module and its *incoming_message_resolver* sub-module. The final output of the hierarchical design is a tree of data-flow graphs, with a total of 8 composite operators (internal nodes) and twenty-six atomic operators (leaf nodes). During translation, these leaf nodes are merged into a single-level data-flow graph automatically and the resultant precedence and timing constraints are used by the static scheduler to generate a run-time static schedule for the time-critical operators. The user interface for the prototype was also developed using the TAE Work Bench. It consists of a total of 14 panels, four of which are shown in Figure 8.

4. LESSONS LEARNED

It took the students a total of about one man-month to complete each of the above projects. They spent about 25% of the time learning and debugging the CAPS tools, 20% of the time analyzing the requirements and developing PSDL specifications for the prototypes, 40% of the time developing Ada codes for the operators and the user interface, 10% of the time debugging and testing the prototype, and 5% of the time documenting the project and writing the final report.

One of the major difficulties encountered by the students was the lack of published user manuals for the CAPS tools, thus making the learning curve steeper than desired. Furthermore, the use of the CAPS tools also revealed several bugs in the Design Database, PSDL Syntax-directed editor, and Static Scheduler, which added extra burden to the students using these tools. Another difficulty encountered by the students was learning how TAE Plus works and how to modify the Ada code generated by the TAE Work Bench to fit the CAPS real-time model. Most of these difficulties can be traced to the fact that CAPS was still under development. Several of its components were not available at the time of the classroom projects. Today, two years later, the tools are much more mature and instructors have gained more experience in the use of course material. We are looking forward to future experiments in combining the teaching and research efforts for improving the quality of education for our graduate students.

The availability of even an experimental version of the computer-aided prototyping tools did help students in visualizing and understanding the effects of real-time constraints. Through the use of CAPS tools, students designing the Patriot Missile Defense System prototype were able to go through the cycle of changing the timing constraints of the operators, translating the PSDL specification into Ada code, compiling the Ada code and executing the prototype in less than 5 minutes. As they varied the maximum execution time and periods of the time-critical operators, they discovered that the Patriot system would fail to intercept the incoming Scud missile due to either missing deadlines when the timing constraints were too tight or inaccurate flight path computations when the timing constraints were too loose. Such experiments would be very difficult, if not impossible, to conduct without the help of computer-aided prototyping tools.

The CAPS tools also facilitate the coordination and management of team work. After completing the top-level design of the C3I station prototype, the students were split into three subgroups, each responsible for the design and development of roughly one-third of the prototype. These subgroups worked independently, interacting with each other only through the CAPS

design database. When the design was completed, the CAPS translator automatically merged all the atomic operators together and generated the drivers and run-time schedules needed to execute the prototype. The final complete prototype has roughly a total of ten thousand lines of Ada code.

5. CONCLUSIONS

It is possible to prototype realistic hard real-time systems in a classroom environment rapidly using of the CAPS tools. The on-line design database helps coordinate and manage team work. The automatic generation of the Ada main program and run-time schedules helps student to experiment with different design options quickly.

Experiences gained from these prototyping exercises have also suggested many possible improvements to future version of CAPS. A new user interface is needed to unify the various tool user interfaces into one and to make the boundaries between different tools transparent to the user. Better integration between the graphic editor and the syntax-directed editor is needed to enforce consistency of the multi-level designs. The current version of Design Database is very primitive and is one of the major bottlenecks in CAPS. A more advanced Design Database based on the revised Design Database Model is needed to support concurrent design effort. Depending on the timing constraints specified in the prototype, the Static Scheduler in CAPS may produce very large periodic static schedules that cause the Ada compiler to fail; an alternate approach to Harmonic Block scheduling is needed. Efforts are currently under way to respond to all of these issues.

6. ACKNOWLEDGMENT

The authors would like to thank V. Berzins, F. Palazzo, A. Wong, L. Williamson, C. Altizer, past and present members of the CAPS research group, and the students in the CS3050, CS3460, CS4500, and CS4530 classes for their contributions to the CAPS research and the prototyping projects.

7. REFERENCES

[1] GAO Report, "Patriot Missile Defense - Software Problem Led to System Failure at Dhahran, Saudi Arabia", GAO/IMTEC-92-26, United States General Accounting Office, Washington, D.C. 20548, Feb. 1992.

[2] J.K. Stankovic and K. Ramamritham, *Tutorial on Hard Real-Time Systems*, IEEE Computer Society Press, Washington, D.C., 1988.

[3] L. Sha, J. Goodenough, "Real-time Scheduling Theory and Ada," *IEEE Computer*, Vol. 23, No. 4, pp. 53-62, April, 1990.

[4] Luqi, "Software Evolution via Rapid Prototyping," *Computer*, vol. 22, pp. 13-25, 1989.

[5] V. Berzins, M. Gray, D. Naumann, "Abstraction-Based Software Development", *CACM*, Vol. 29, No. 5, pp.402-415, May 1986.

[6] Luqi, M. Shing, "CAPS - A Tool for Real-Time System Development and Acquisition.", *Naval Research Reviews*, Vol. XLIV, pp. 12-16, 1992.

[7] V. Berzins, Luqi, *Software Engineering with Abstractions*, Addison-Wesley, ISBN 0-201-08004-4, 1991.

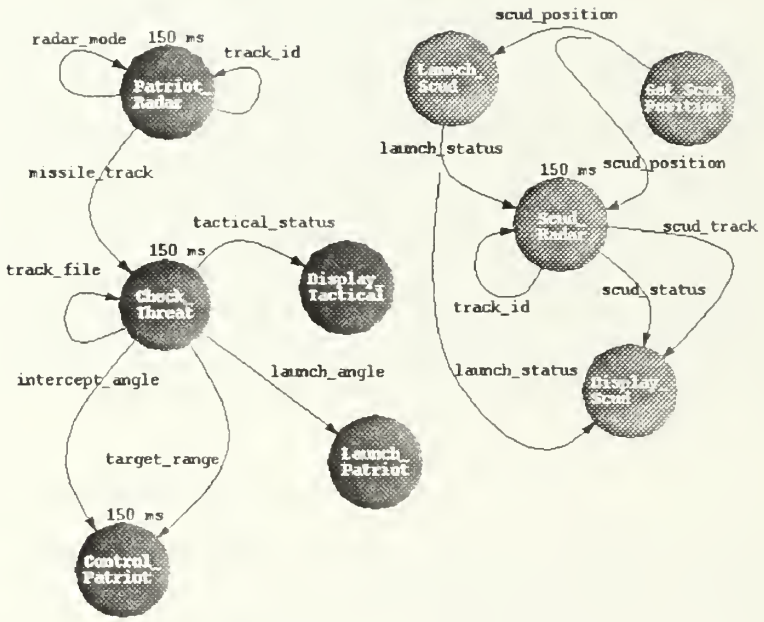
[8] Luqi, R. Steigerwald, G. Hughes, F. Naveda, and V. Berzins, "CAPS as Requirements Engineering Tool," *Proc. Requirements Engineering and Analysis Workshop*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1991.

[9] Luqi, V. Berzins, and R.T. Yeh, "Prototyping Language for Real-Time Software," *IEEE Transactions on Software Engineering*, vol. 14, pp. 1409-1423, 1988.

[10] P. Barnes, "Using TAE Plus to Illustrate Hard Real-Time System Prototypes," *TAE Plus Newsletter*, Summer 1992.

[11] Luqi, "Computer-Aided Prototyping for Command-And-Control Systems using CAPS," *IEEE Software*, pp. 56-67, Jan 1992.

```
OPERATOR patriot
SPECIFICATION
    DESCRIPTION
      {Patriot Missile Defense System simulation.
       Simulates Patriot intercepting a Scud missile.}
END
IMPLEMENTATION
    GRAPH
```




```
    DATA STREAM
      -- Type declarations for the data streams in the graph go here.
    CONTROL CONSTRAINTS
      OPERATOR Patriot_Radar
       PERIOD 800 MS

      OPERATOR Scud_Radar
       PERIOD 800 MS

      OPERATOR Display_Tactical
       TRIGGERED BY SOME
         tactical_status

      OPERATOR Control_Patriot
       TRIGGERED BY ALL
         intercept_angle,
         target_range

      OPERATOR Launch_Patriot
       TRIGGERED BY ALL
         launch_angle

      OPERATOR Check_Threat
       PERIOD 800 MS
       OUTPUT
         intercept_angle
         IF NOT (intercept_angle = 0.0)
       OUTPUT
         target_range
         IF NOT (intercept_angle = 0.0)
       OUTPUT
         launch_angle
         IF NOT (launch_angle = 0.0)

END
```

Figure 3. PSDL Specification of the Patriot Missile Defense System

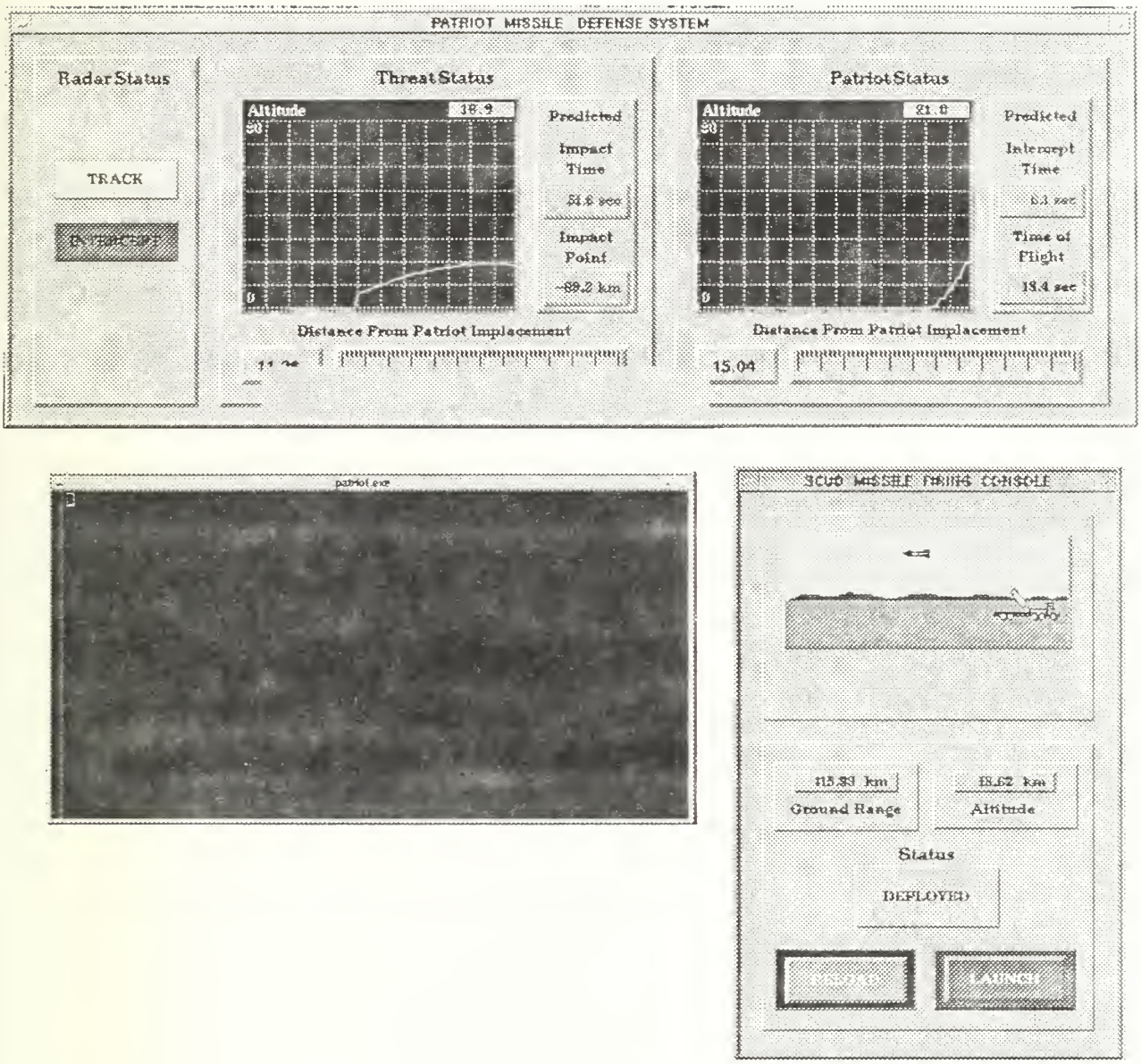
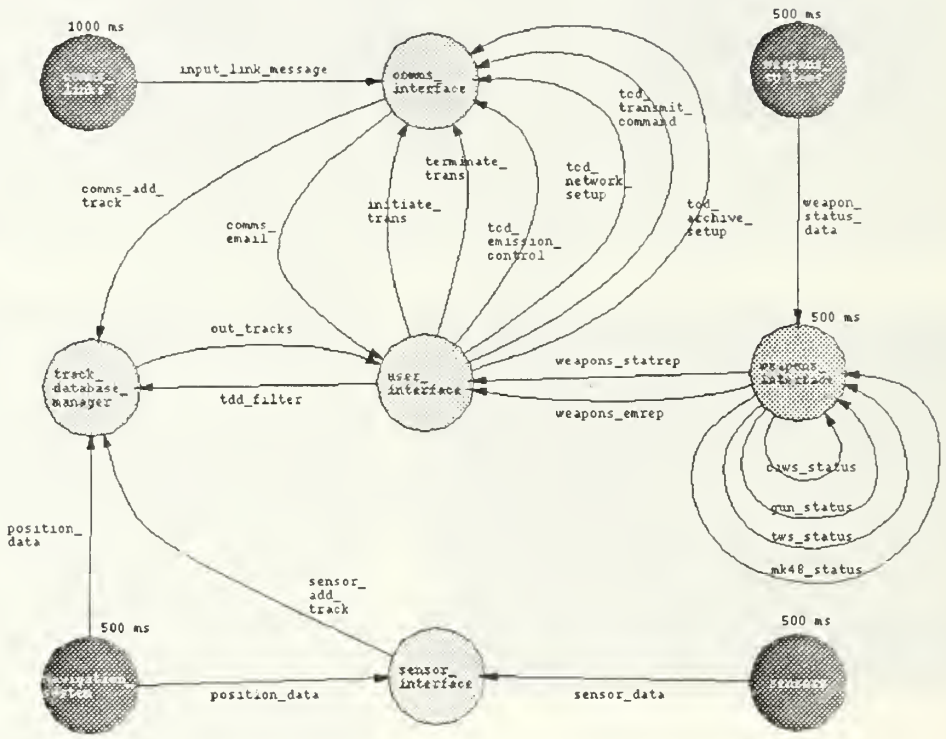


Figure 4. User Interface for the Patriot Missile Defense System Prototype

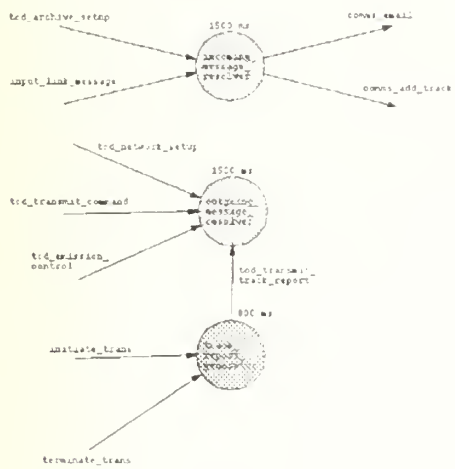
```
OPERATOR c3i_system
SPECIFICATION
    DESCRIPTION
      {This module implements a simplified version of
       a generic C3I workstation.}
END
IMPLEMENTATION
    GRAPH
```




```
    DATA STREAM
      -- Type declarations for the data streams in the graph go here.
    CONTROL CONTRAINTS
      OPERATOR comms_links
        PERIOD 30000 MS

      OPERATOR navigation_system
        PERIOD 30000 MS

      OPERATOR sensors
        PERIOD 30000 MS

      OPERATOR weapons_systems
        PERIOD 30000 MS

      OPERATOR weapons_interface
        TRIGGERED BY SOME
          weapon_status_data
        MINIMUM CALLING PERIOD 2000 MS
        MAXIMUM RESPONSE TIME 3000 MS
        OUTPUT
          weapons_emrep
          IF weapon_status_data.status =
            damaged
          OR weapon_status_data.status =
            service_required
          OR weapon_status_data.status =
            out_of_ammunition
END
```

Figure 5. PSDL Specification of the C3I_system

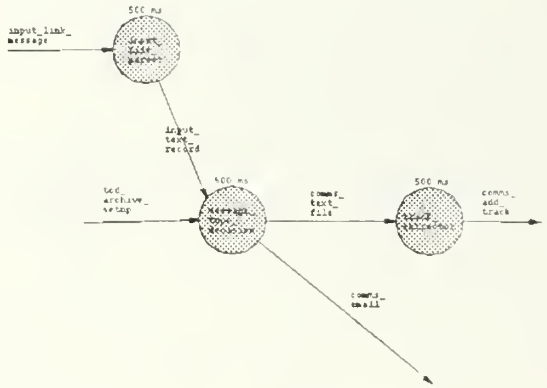
```
OPERATOR comms_interface
SPECIFICATION
    INPUT
      tcd_transmit_command : transmit_command,
      tcd_archive_setup : archive_setup,
      tcd_network_setup : network_setup,
      tcd_emission_control : emissions_control_command,
      terminate_trans : BOOLEAN,
      initiate_trans : initiate_transmission_sequence,
      input_link_message : filename
    OUTPUT
      comms_email : filename,
      comms_add_track : add_track_tuple
    DESCRIPTION
      (This operator is responsible for processing incoming
       and outgoing messages as well as producing periodic
       track report and format translating)
END
IMPLEMENTATION
    GRAPH
```



```
    DATA STREAM
      -- Type declarations for the data streams in the graph go here.
    CONTROL CONSTRAINTS
      OPERATOR track_report_generator
        TRIGGERED IF
          NOT terminate_trans
        PERIOD 30000 MS
END
```

Figure 6. PSDL Specification for the comms_interface module

```
OPERATOR incoming_message_resolver
SPECIFICATION
    INPUT
      tcd_archive_setup : archive_setup,
      input_link_message : filename
    OUTPUT
      comms_add_track : add_track_tuple,
      comms_email : filename
    MAXIMUM EXECUTION TIME 1500 ms
    DESCRIPTION
      {This operator processes the incoming messages.}
END
IMPLEMENTATION
    GRAPH
```



```
    DATA STREAM
      -- Type declarations for the data streams in the graph go here.
    CONTROL CONSTRAINTS
      OPERATOR input_file_parser
        TRIGGERED BY SOME
          input_link_message

      OPERATOR message_type_decision
        TRIGGERED BY SOME
          input_text_record
        OUTPUT
          comms_text_file
          IF comms_text_file.archive
        OUTPUT
          comms_email
          IF  NOT input_text_record.is_track

      OPERATOR track_extractor
        TRIGGERED IF
          comms_text_file.is_track
END
```

Figure 7. PSDL Specification of the incoming_message_resolver module

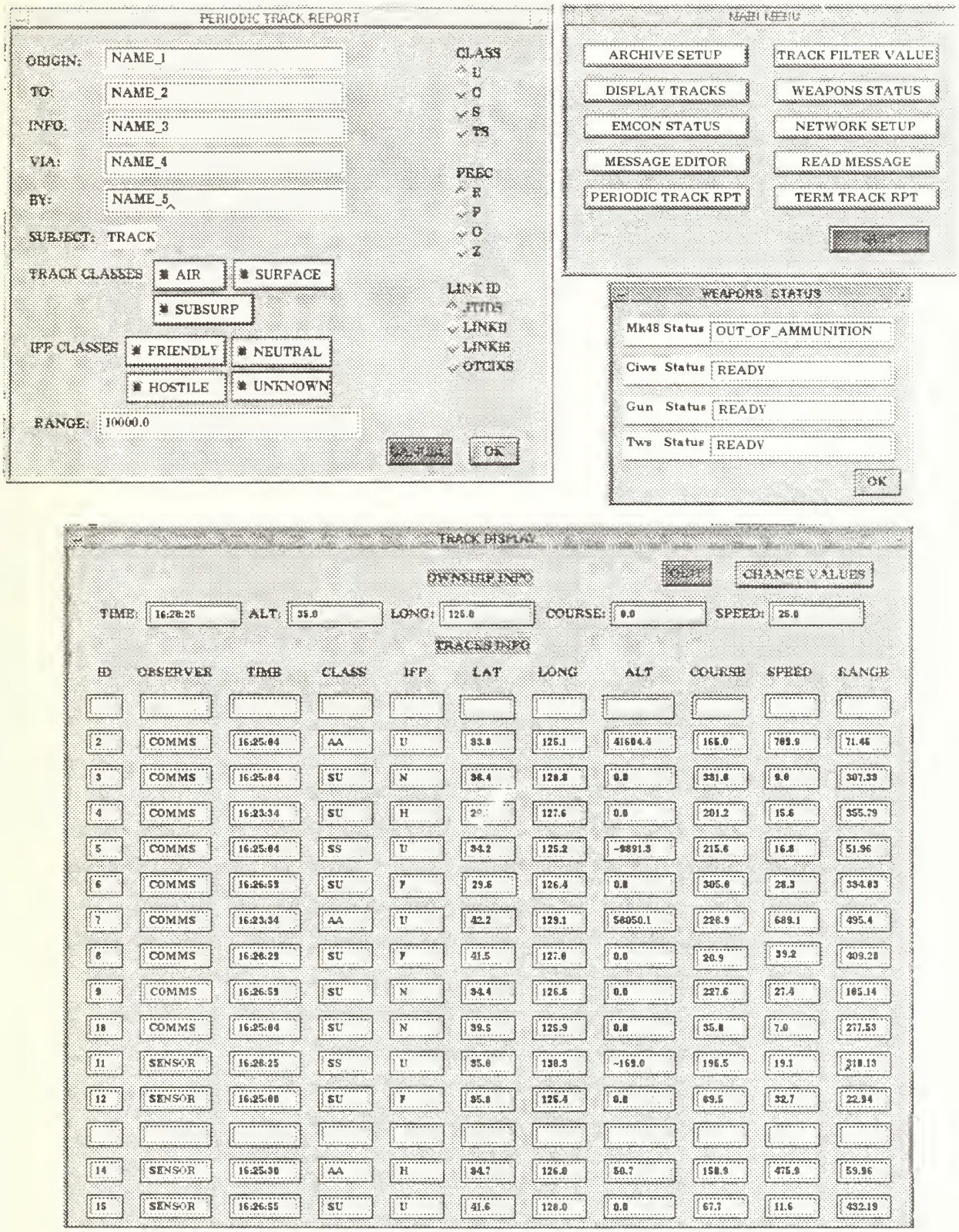


Figure 8. User Interface for the C3I_system

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943-5100	2
Office of Research Administration Code 08 Naval Postgraduate School Monterey, CA 93943-5100	1
Chief of Naval Research 800 N. Quincy Street Arlington, VA 22302-0268	1
Center for Naval Analysis 4401 Ford Avenue Alexandria, VA 22302-0268	1
Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, D.C. 20350-2000	1
Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	2
Dr. Luqi Computer Science Department, Code CSLq Naval Postgraduate School Monterey, CA 93943-5100	10
Dr. Mantak Shing Computer Science Department, Code CSSh Naval Postgraduate School Monterey, CA 93943-5100	10

CPT Patrick Barnes
PSC 20, P. O. Box 3063
APO AE 09260 2

CDR Gary Hughes
Computer Science Department, Code CSHu
Naval Postgraduate School
Monterey, CA 93943-5100 2

Dr. Thomas Wu
Computer Science Department, Code CSWq
Naval Postgraduate School
Monterey, CA 93943-5100 1